Formal Methods in System Design manuscript No.
(will be inserted by the editor)

Hybrid Systems: From Verification to Falsification by Combining Motion Planning and Discrete Search

Erion Plaku · Lydia E. Kavraki · Moshe Y. Vardi

Received: date / Revised version: date

Abstract We propose `HyDICE`, Hybrid DIscrete Continuous Exploration, a multi-layered approach for hybrid-system falsification that combines motion planning with discrete search and discovers safety violations by computing witness trajectories to unsafe states. The discrete search uses discrete transitions and a state-space decomposition to guide the motion planner during the search for witness trajectories. Experiments on a nonlinear hybrid robotic system with over one million modes and experiments with an aircraft conflict-resolution protocol with high-dimensional continuous state spaces demonstrate the effectiveness of `HyDICE`. Comparisons to related work show computational speedups of up to two orders of magnitude.

Keywords Hybrid system · Safety properties · Robot motion planning · Discrete search · Sampling-based planning · Decomposition · Nonlinear dynamics

1 Introduction

1.1 Hybrid Systems and Verification of Safety Properties

Hybrid systems play an increasingly important role in transportation networks as part of sophisticated embedded controllers used in the automotive industry and air-traffic management, or in manufacturing processes, robotics, and even medicine and biology as part of medical devices monitoring health conditions [6, 24, 29, 42, 43, 54]. A hybrid

Work supported in part by NSF CNS 0615328 (EP, LK, MV), NSF 0713623 (EP, LK), a Sloan Fellowship (LK), and NSF CCF 0613889 (MV). Equipment supported by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice, AMD, and Cray.

A preliminary version of this work was published in the Proceedings of the 19th International Conference on Computer Aided Verification (CAV 2007). Lecture Notes in Computer Science, eds. W. Damm and H. Hermanns, vol. 4590, pp. 468–481. This work contains substantial improvements to the overall computational method introduced in the preliminary work and new experiments that were not included in the preliminary work.

Erion Plaku · Lydia E Kavraki · Moshe Y. Vardi
Department of Computer Science, Rice University,
Houston, TX 77005 USA. E-mail: {plakue, kavraki, vardi}@cs.rice.edu

system is a formal model that combines discrete and continuous dynamics. Continuous dynamics are associated with each mode, and discrete logic determines how to switch between modes. A hybrid system may model air traffic control, where the modes correspond to the cruising of the planes and the discrete logic models conflict-resolution protocols. As another example, a hybrid system may model a vehicle whose underlying dynamics varies discretely depending on terrain conditions.

As hybrid systems are often part of devices operating in safety-critical situations, the verification of safety properties becomes increasingly important. A hybrid system is considered safe if unsafe states cannot be reached starting from initial safe states.

The hybrid-system verification problem has traditionally been formulated as a reachability analysis on the state space of the hybrid system. In the forward reachability formulation, safety verification is equivalent to showing that the set of states reachable from the initial states does not intersect the set of unsafe states. In the backward reachability formulation, safety is guaranteed by showing that the set of states that can reach an unsafe state does not intersect the initial set of states.

Over the years a rich theory has been developed for this problem as well as numerous methods [1, 25, 26, 36, 39, 48]. Initial approaches included enumeration and symbolic methods originally developed for discrete systems [14]. Tools such as KRONOS [55] and UPPAAL [5] have been used for the verification of real-time hardware and software, and HyTech [27] has been used for the verification of hybrid systems with linear dynamics.

Research has also focused on abstraction methods that make verification more amenable to analysis by constructing a simplified model that simulates the original system [2,3,12,23,49]. The simplified model is usually obtained by eliminating variables that do not influence safety properties, mapping each domain to a smaller domain, or constructing finite-state models that group states that satisfy the same predicates.

Alternative methods have also been developed that approximate the reachable set [1, 4, 8, 51–53]. Tools such as d/dt [4], Checkmate [51], VeriSHIFT [8] use polyhedra or ellipsoids to overapproximate the reachable set, and other tools use level sets to compute convergent approximations [53].

1.2 From Verification to Falsification

Unfortunately, even for safety properties where verification is equivalent to reachability checking, decidability holds only for hybrid systems with simple continuous dynamics (essentially some types of linear dynamics) [1, 26, 40, 53]. In light of these theoretical results, it is no surprise that the most efficient complete algorithms for hybrid-system verification have a single- or double-exponential dependency on the dimension of the state space and are generally limited in practicality to hybrid systems with up to six dimensions, simple dynamics, and few or no input controls [1, 40, 53].

These hardness theoretical results underscore the need for the development of alternative methods that perhaps satisfy weaker forms of completeness, but can handle more general hybrid systems. In fact, recent computational methods developed in [7,9,19,30,32,41], even though unable to determine that a hybrid system is safe, are capable of handling nonlinear hybrid systems and finding unsafe behaviors when such systems are unsafe.

In essence, the focus in these recent approaches shifts from *verification* to *falsification*, which often is the main focus of model checking in industrial applications [15].

Falsification (see [7,32,41]) studies the following problem: *Can a hybrid-system witness trajectory be produced from a safe state to an unsafe state when such trajectories exist?*

The main contribution of this work is the development of an efficient computational method for hybrid-system falsification that offers significant computational speedups of up to two orders of magnitude over related work. When a hybrid system is safe, it may not be possible to prove that unsafe states are unreachable. Such an approach trades completeness for the ability to discover safety violations for complex hybrid systems with nonlinear dynamics and input controls that current verification methods cannot handle.

1.3 Combining Motion Planning and Discrete Search for the Falsification of Safety Properties of Hybrid Systems with Nonlinear Dynamics

This work approaches hybrid-system falsification from a robotics perspective. Initially, we exploit the insight that hybrid-system falsification is in many respects related to robot motion planning, which is a search problem for a witness trajectory that satisfies certain invariants, such as ensuring that the robot motion respects kinodynamic constraints and avoids collision with obstacles [11,37]. While in motion planning the search takes place in a continuous space, in hybrid-system falsification the search for a witness trajectory takes place in a space consisting of discrete and continuous components.

The connection between hybrid-system falsification and motion planning becomes deeper when we consider state-of-the art motion planning as the starting point for searching the continuous state spaces of a hybrid system. Recent progress in sampling-based motion planning has made it possible to efficiently find witness trajectories even for high-dimensional and nonlinear continuous systems (e.g., `PRM` [31], `RRT` [38], `EST` [28,50], `PDST` [35], `DSLX` [45], and others surveyed in [11,37]). These motion planners typically search the continuous state space by incrementally extending feasible trajectories in a tree-like fashion from initial states toward unsafe states. Recently, `RRT`-based methods have also been used for the falsification of safety properties of nonlinear hybrid systems with few modes [7, 9, 19, 32, 41].

Departing from traditional robot motion planning, we introduce a discrete-search component to our work that is responsible for managing the potentially huge number of modes and discrete transitions of a hybrid system. The contribution of this work is the development of a multi-layered framework for hybrid-system falsification that effectively combines sampling-based motion planning with discrete search. The motivation and many of our design decisions come from our earlier work [45–47]. In [47] we use discrete search to obtain a sequence of discrete transitions that guides the generation of motions for a hybrid robotic system with 10–30 modes and mostly linear dynamics. In [45] we show that traditional motion-planning problems can be solved more efficiently by combining sampling-based motion planning with discrete search over an artificially imposed decomposition of the environment on which the robot moves (which in general can be regarded as a projection of its state space). In [46] we show that combination of motion planning and discrete search is also promising for hybrid-system falsification.

The work in this article combines and extends ideas in [45–47] to obtain an effective falsification method for hybrid systems. As a result, while our previous work [46] could handle hybrid systems with up to ten thousand modes, this work can handle hybrid systems with over a million modes and nonlinear dynamics associated with each mode. In addition, while our previous work [46] focused on hybrid systems with low-dimensional continuous spaces, this work shows the effectiveness of the proposed approach even for hybrid systems with high-dimensional continuous spaces. The proposed method, `HyDICE`, uses discrete transitions and a decomposition of the continuous state spaces into regions to construct a search graph that provides a simplified layer to the hybrid-system falsification problem. Vertices of the graph correspond to decomposition regions, while edges correspond to adjacent decomposition regions or decomposition regions that are connected by a discrete transition. The discrete-search component of `HyDICE` obtains from this graph at each iteration a high-level plan, called a *lead*, that guides the motion planner in the search for a witness trajectory. Each lead corresponds to a sequence of decomposition regions and discrete transitions that start at a decomposition region associated with an initial safe state and end at a decomposition region associated with an unsafe state. Fig. 1 provides an illustration[1]. Among the combinatorially large number of such sequences, the discrete-search component of `HyDICE` computes at each iteration a lead that is estimated to be a useful search direction for finding a witness trajectory. The search inside the decomposition regions associated with the lead is based on a state-of-the-art sampling-based motion planner [45]. The motion planner sample states inside the decomposition regions, connects states associated with the same mode with simple continuous trajectories, and connects states associated with different modes by interleaving continuous trajectories with discrete transitions, as shown in Fig. 2. A witness trajectory is then found when the motion planner succeeds in connecting a safe state to an unsafe state. Coverage estimation is fed back from the motion planner to the discrete search in order to improve the lead in the next iteration. This interaction between the motion planner and discrete search, illustrated in Fig. 2, is crucial for the efficiency of `HyDICE`.

In contrast to previous work [32, 41], as shown later in this article, `HyDICE` is well-suited for systems with many modes. Experimental validation is provided by using `HyDICE` for the falsification of safety properties of a hybrid robotic system with over one million modes, and nonlinear dynamics and input controls associated with each mode. An additional benchmark, based on aircraft conflict-resolution protocols, demonstrates the effectiveness of `HyDICE` in the case of hybrid systems with high-dimensional continuous state spaces. As indicated by the experiments, the tight integration of discrete search and motion planning enables `HyDICE` to be up to two orders of magnitude faster than other related methods.

The rest of the article is as follows. The hybrid-system model, hybrid-system falsification problem, and the related motion-planning problem are described in Section 2. Description of `HyDICE` is given in Section 3. Experiments and results are presented in Section 4. The article concludes in Section 5 with a discussion.

2 Preliminaries

This section defines hybrid automata, the hybrid-system falsification problem, and the related motion-planning problem.

[1] Figures in this work are better viewed in color.

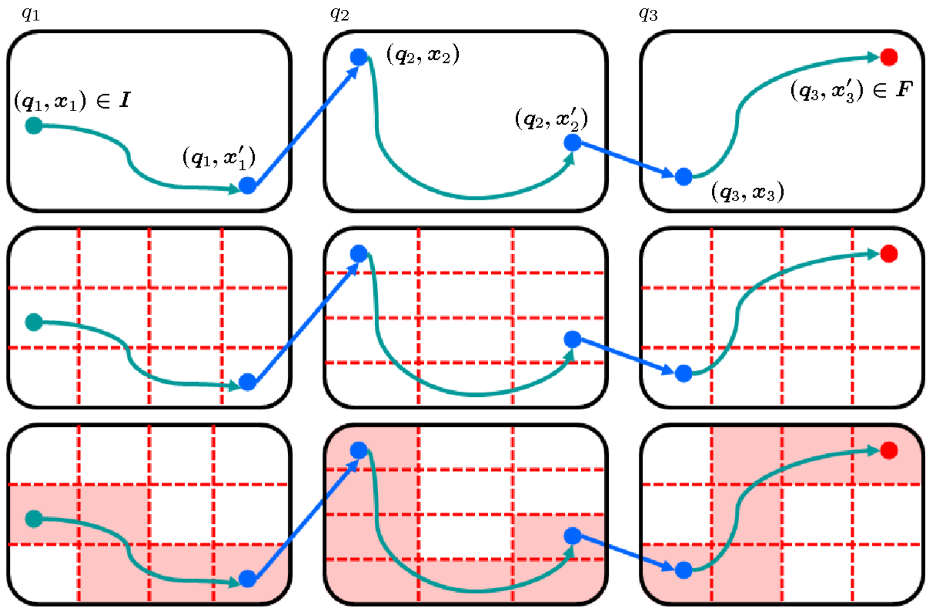

Fig. 1 (top) Example of a witness hybrid-system trajectory from a safe state (q_1, x_1) to an unsafe state (q_3, x_3). The witness trajectory consists of continuous trajectories, shown as curved green arrows, interleaved with discrete transitions, shown as straight blue arrows. (middle) The dotted red regions show a decomposition of the continuous state spaces associated with the different modes q_1, q_2, q_3 of the hybrid system. (bottom) The shaded decomposition regions show a lead, a sequence of decomposition regions from the safe to the unsafe state that is estimated to be a useful search direction for finding a witness trajectory.

2.1 Hybrid Automata and Hybrid-System Falsification Problem

In this work, hybrid systems are modeled by hybrid automata [1].

Definition 1 A hybrid automaton is a tuple

$$H = (S, Inv, E, G, J, U, f, I, F),$$

where

- $S = Q \times X$ is the Cartesian product of the discrete and continuous state spaces;
- Q is a discrete and finite set;
- X maps each mode to the corresponding continuous state space, i.e., $q \overset{X}{\mapsto} X_q$, where $X_q \subset \mathbb{R}^{\dim(X_q)}$ is the continuous state space associated with $q \in Q$;
- Inv maps each mode to the corresponding continuous invariant, i.e., $q \overset{Inv}{\mapsto} Inv_q$, where $Inv_q \subseteq X_q$ represents the domain of the continuous variables associated with $q \in Q$;
- $E \subseteq Q \times Q$ is the set of discrete transitions between modes;
- G maps discrete transitions to guard conditions, i.e., $(q_i, q_j) \overset{G}{\mapsto} G_{(q_i,q_j)}$, where $G_{(q_i,q_j)} \subseteq X_{q_i}$ is the guard condition associated with $(q_i, q_j) \in E$;
- J maps discrete transitions to reset functions, i.e., $(q_i, q_j) \overset{J}{\mapsto} J_{(q_i,q_j)}$, where $J_{(q_i,q_j)} : G_{(q_i,q_j)} \to X_{q_j}$ is the reset function associated with $(q_i, q_j) \in E$;

- U maps each mode to the corresponding set of input controls, i.e., $q \overset{U}{\mapsto} U_q$, where $q \in Q$ and $U_q \subseteq \mathbb{R}^{\dim(U_q)}$;
- f maps each mode to the function that describes the associated continuous dynamics, i.e.,$q \overset{f}{\mapsto} f_q$, where $f_q : X_q \times U_q \rightarrow \dot{X}_q$ determines the continuous dynamics associated with $q \in Q$, and $\dot{X}_q$ is the tangent space of X_q;
- $I \subset S$ is the set of initial states; and
- $F \subset S$ is the set of unsafe states.

The state of the hybrid automaton is a tuple $(q, x) \in S$ that describes both the discrete and the continuous components. The invariant, $Inv_q \subseteq X_q$, associated with each mode $q \in Q$, represents the domain of the continuous variables $x \in X_q$. The set E describes which transitions are possible from one mode to another. A discrete transition $(q_i, q_j) \in E$ occurs when the corresponding guard condition $G_{(q_i,q_j)}$ is satisfied. The state of the system is then reset according to the reset function $J_{(q_i,q_j)}$. The continuous dynamics of the system in each $q \in Q$ is governed by a set of differential equations $f_q : X_q \times U_q \rightarrow \dot{X}_q$. In this work, each $X_q \in X$ includes derivatives of different orders, e.g., velocity and acceleration of a vehicle, and thus f_q is nonlinear. The function f_q has the form $f_q(x, u)$, where the input $u \in U_q$ could represent controls, nondeterminism, uncertainties, disturbances from the environment, or actions of other systems.

A hybrid-system trajectory consists of one or more continuous trajectories interleaved with discrete transitions. A hybrid system is considered unsafe if a trajectory is found that reaches an unsafe state starting from an initial safe state. More precisely, the problem statement is as follows.

Definition 2 A state $s = (q, x) \in S$, a time $T \geq 0$, and an input control $u \in U_q$, define a valid continuous trajectory $\Psi_{s,u,T} : [0, T] \rightarrow X_q$ when

- $x = \Psi_{s,u,T}(0)$;
- $\Psi_{s,u,T}(t) \in Inv_q$, for $t \in [0, T]$;
- $\Psi_{s,u,T}(t) \in Inv_q - \{G_{(q,q')} : (q, q') \in E\}$, for $t \in [0, T)$; and
- $\dot{\Psi}_{s,u,T}(t) = f_q(\Psi_{s,u,T}(t), u)$, for $t \in [0, T]$.

For any state $s = (q, x) \in S$, define

$$\chi(q.x) = \begin{cases} \chi\left(q', J_{(q,q')}(x)\right), & x \in G_{(q,q')} \ \mathit{for\ some}\ (q, q') \in E, \\ (q, x), & \mathit{otherwise.} \end{cases}$$

The hybrid-system trajectory $\Upsilon_{s,u,T} : [0, T] \rightarrow S$, defined as

$$\Upsilon_{s,u,T}(t) = \begin{cases} (q, \Psi_{s,u,T}(t)), & 0 \leq t < T, \\ \chi(q, \Psi_{s,u,T}(t)), & t = T, \end{cases}$$

ensures that discrete transitions at time T, if they occur, are followed.

The continuous trajectory $\Psi_{s,u,T}$ is thus obtained by applying the input control u to the state s for a duration of T units of time. Moreover, $\Psi_{s,u,T}$ never reaches a guard condition during the time interval $[0, T)$ and each state of $\Psi_{s,u,T}$ satisfies the invariant. The trajectory $\Upsilon_{s,u,T}$ is similar to $\Psi_{s,u,T}$, but, unlike $\Psi_{s,u,T}$, $\Upsilon_{s,u,T}$ follows the discrete transitions at time T when they occur.

We note that in the hybrid-system benchmarks used in this work the discrete transitions are considered urgent, i.e., a discrete transition is immediately taken once a guard condition is satisfied. There is however no inherent limitation of `HyDICE` in dealing with non-urgent discrete transitions. When discrete transitions are non-urgent, enabled discrete transitions could be taken nondeterministically with some probability or taken only when the invariant is invalid or a combination of both.

Definition 3 The extension of a trajectory $\Phi : [0, T] \to S$ by applying to $\Phi(T)$ the input control $u' \in U$ for a duration of time $T' > 0$ is written as

$$\Phi \circ (u', T'),$$

and it is another trajectory $\Xi : [0, T + T'] \to S$ defined as

$$\Xi(t) = \begin{cases} \Phi(t), & t \in [0, T], \\ \Upsilon_{\Phi(T),u',T'}(t - T), & t \in (T, T + T']. \end{cases}$$

The trajectory $\Phi \circ (u', T')$ thus denotes the hybrid-system trajectory that is obtained by applying the input control u' to the last state of Φ for a duration of T' units of time and following all the discrete transitions that may occur at time $T + T'$.

Definition 4 (Problem Statement) Given a hybrid automaton H, find a sequence $u_1, u_2, \dots, u_k$ of input controls and a sequence $T_1, T_2, \dots, T_k$ of time durations, such that the trajectory $\mathcal{W} : [0, T] \to S$ defined as

$$\mathcal{W} \overset{def}{=} \Upsilon_{s_{\text{safe}},u_1,T_1} \circ (u_2, T_2) \circ \cdots \circ (u_k, T_k),$$

reaches an unsafe state, i.e., $\mathcal{W}(T) \in F$, where $T = T_1 + \cdots + T_k$ and $s_{\text{safe}} \in I$.

2.2 Motion-Planning Problem

The motion-planning problem consists of finding a trajectory for a robotic system from an initial state to a final state, such that the trajectory satisfies kinodynamic and other constraints on the robot motion, e.g., bounds on velocity and acceleration, collision avoidance. In an abstract formulation, the motion-planning problem is closely related to the hybrid-system falsification problem, as evidenced by the following definition:

Definition 5 The motion-planning problem is a tuple

$$MP = (X, Inv, U, f, I, F),$$

where

- $X \subset \mathbb{R}^{\dim(X)}$ is the continuous state space;
- $Inv \subset X$ is the invariant set representing the domain of the continuous variables;
- $U \subseteq \mathbb{R}^{\dim(U)}$ is the set of input controls;
- $f : X \times U \to \dot{X}$ determines the continuous dynamics, and $\dot{X}$ is the tangent space of X;
- $I \subset S$ is the set of initial states; and
- $F \subset S$ is the set of final states.

A solution to the motion-planning problem is a witness trajectory from a state $s' \in I$ to a state $s'' \in F$, such that each state in this trajectory satisfies the invariant Inv.

Algorithm 1 A search-tree framework for finding a witness trajectory

```
Input: $H = (S, Inv, E, G, J, f, U, I, F)$: hybrid system
       $t_{\max} \in \mathbb{R}$: upper bound on overall computation time
Output: A witness trajectory or FAILURE if no witness trajectory is found

1: StartClock
2: $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$; $V_\mathcal{T} \leftarrow \{s_{\text{safe}}\}$; $E_\mathcal{T} \leftarrow \emptyset$
3: while ElapsedTime $< t_{\max}$ do
4:    $s \leftarrow$ SelectStateFromSearchTree($H, \mathcal{T}$)
5:    $s_{\text{new}} \leftarrow$ ExtendSearchTree($H, \mathcal{T}, s$)
6:    $V_\mathcal{T} \leftarrow V_\mathcal{T} \cup \{s_{\text{new}}\}$; $E_\mathcal{T} \leftarrow E_\mathcal{T} \cup \{(s, s_{\text{new}})\}$
7:    if $s_{\text{new}} \in F$ then
8:       return WitnessTrajectory($\mathcal{T}, s_{\text{new}}$)
9: return FAILURE
```

The invariant Inv represents different constraints imposed on the states of the systems and indicates which states satisfy those constraints. The invariant is usually specified implicitly as $Inv = \{x : x \in X \wedge \text{val}(x) = 1\}$, where the function val : $X \rightarrow \{0, 1\}$ indicates which state is valid.

A comparison of the hybrid automaton in Definition 1 and the motion-planning problem in Definition 5 reveals the similarities between them. In fact, the motion-planning problem corresponds to a hybrid automaton that has only one mode and no discrete transitions. As it will be explained in Section 3, `HyDICE` takes advantage of precisely this similarity to effectively search the continuous state spaces associated with the modes of a hybrid system.

3 `HyDICE`

A preliminary version of `HyDICE` has appeared in [46]. This section provides a detailed description of `HyDICE` and emphasizes the extensions aimed at improving the motion planner and the interplay between the motion planner and the discrete search. As a result of these extensions, as shown in Section 4, the overall computational efficiency of the method improves significantly over [46].

Throughout execution, `HyDICE` maintains an internal data structure, which is a tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$. A vertex $s \in V_\mathcal{T}$ is a state in S, while an edge $(s', s'') \in E_\mathcal{T}$ indicates that a hybrid-system trajectory connects $s' \in S$ to $s'' \in S$. Initially $\mathcal{T}$ contains a safe state $s_{\text{safe}} \in I$ as its root and has no edges, i.e., $V_\mathcal{T} = \{s_{\text{safe}}\}$ and $E_\mathcal{T} = \emptyset$. The search for a witness trajectory proceeds in an iterative fashion. At each iteration, $\mathcal{T}$ is extended by adding a new vertex to $V_\mathcal{T}$ and a new edge to $E_\mathcal{T}$. The search terminates successfully when an unsafe state $s_{\text{unsafe}} \in F$ is added to $\mathcal{T}$. A witness trajectory is then constructed by concatenating the hybrid-system trajectories associated with the tree edges that connect s_{safe} to s_{unsafe}. Otherwise, the search continues until an upper bound on the computation time is exceeded. Algorithm 1 provides pseudocode for this general search-tree framework.

3.1 Extending the Search-Tree Framework

The success of the search-tree framework in Algorithm 1 depends on the ability of the method to quickly extend $\mathcal{T}$ along those directions that can facilitate the construction of a witness trajectory. `HyDICE`, as explained next, uses the discrete transitions of the hybrid system and a state-space decomposition to estimate such directions.

Note that a witness trajectory consists of several continuous trajectories interleaved with discrete transitions, as illustrated in Fig. 1. Each continuous trajectory corresponds to a local connection, i.e., a trajectory between two continuous states associated with the same mode, while each discrete transition occurs when some guard condition is satisfied. In order to construct a witness trajectory, it suffices to identify states where discrete transitions of a witness trajectory occur and use local connections to obtain the continuous trajectories associated with a witness trajectory. Assume for the moment that such computational methods are available, i.e.,

Transitions: Returns a sequence of states $s_{\text{safe}} = (q_1, x_1)$, (q_1, x_1'), (q_2, x_2), (q_2, x_2'), $\cdots$, $(q_n, x_n) = s_{\text{unsafe}}$, where $(q_i, x_i') \in G_{(q_i, q_{i+1})}$ and $x_{i+1} = J_{(q_i, q_{i+1})}(x_i')$.

ConnectSameQ: Given $q \in Q$ and $x', x'' \in Inv_q$, the local connection method returns a continuous trajectory that connects (q, x') to (q, x'').

A witness trajectory can then be constructed by first invoking Transitions and then using ConnectSameQ to connect each (q_i, x_i) to (q_i, x_i') with a continuous trajectory. Observe that in each case ConnectSameQ is solving the motion-planning problem defined in Section 2. `HyDICE` takes advantage of this observation and bases ConnectSameQ on a state-of-the-art motion planner, as described in Section 3.3.

Note that it is in general challenging for the Transitions method to identify states where discrete transitions occur, since witness trajectories are not known a priori. It is however possible to identify sequences of discrete transitions $q_1, q_2, \ldots, q_n$, $(q_i, q_{i+1}) \in E$, from a mode $q_1 = q_{\text{safe}}$ associated with a safe state to a mode $q_n = q_{\text{unsafe}}$ associated with an unsafe state. In fact, these sequences of discrete transitions correspond to paths in the graph (Q, E) of the discrete transitions of the hybrid system.

The objective of `HyDICE` is then to focus the search inside the continuous state spaces associated with these discrete transitions. In particular, the motion-planning component of `HyDICE` attempts to extend $\mathcal{T}$ from states associated with (q_i, X_{q_i}) to states associated with $(q_i, G_{(q_i, q_{i+1})})$, thus enabling discrete transitions to states associated with $(q_{i+1}, X_{q_{i+1}})$. In this way, a sequence of discrete transitions from q_{safe} to q_{unsafe} provides a general direction for extending $\mathcal{T}$ that could potentially facilitate the construction of witness trajectories.

Taking this approach a step further, `HyDICE` also introduces a decomposition of each continuous state space X_q, $q \in Q$, into different regions. Such decomposition has been shown quite effective in increasing the computational efficiency for searching a continuous state space [45]. Moreover, similar to the observation made earlier, when imposing such decomposition, each witness trajectory now passes through a sequence of decomposition regions that starts at a decomposition region associated with $s_{\text{safe}} \in I$ and ends at a decomposition region associated with $s_{\text{unsafe}} \in F$, as illustrated in Fig. 1. Therefore, such a sequence of decomposition regions, which is referred to as a *lead* and described in detail in Section 3.2, provides a potentially useful direction for extending $\mathcal{T}$ during the search for a witness trajectory, as illustrated in Fig. 2.

Interplay of Discrete Search and Motion Planning: Since the number of possible leads could be combinatorially large, `HyDICE` employs a discrete-search component to obtain at each iteration a general direction that is estimated to be useful for extending the search tree $\mathcal{T}$ in order to facilitate the construction of a witness trajectory. The

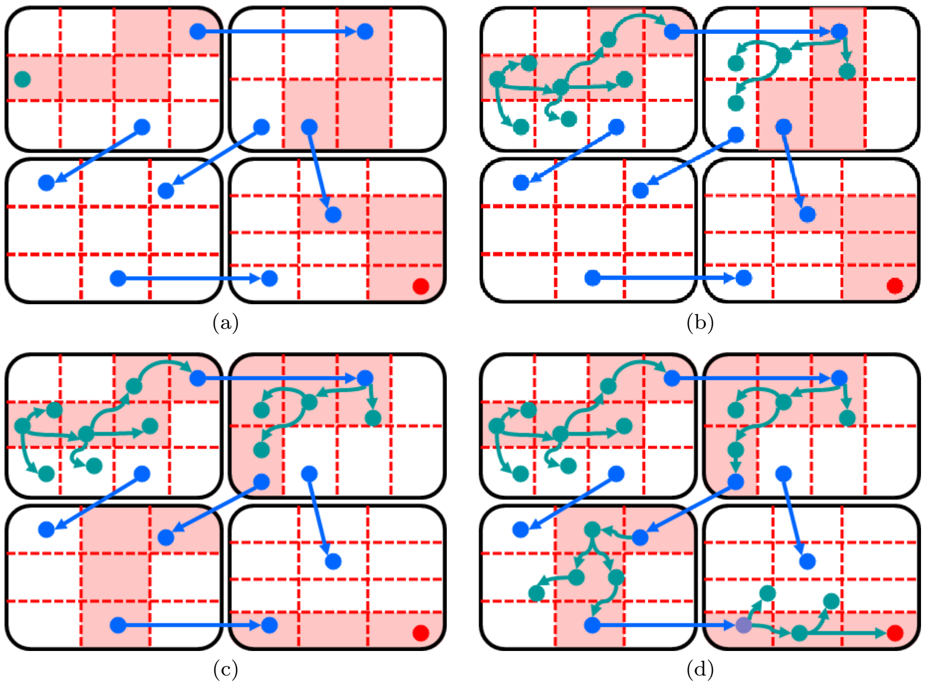

Fig. 2 Illustration of the interplay between the discrete-search and motion-planning components of `HyDICE`. (a) The discrete search computes a lead. (b) The motion planner extends the search tree along the decomposition regions specified by the lead by adding new vertices and edges to the search tree from these or neighboring decomposition regions. (c) A new lead is computed by the discrete search to reflect the current growth of the search tree. (d) The motion planner again extends the search tree using the current lead as a guide. This time, the search tree reaches an unsafe state and thus a witness trajectory is found.

motion-planning component of `HyDICE` extends the search tree $\mathcal{T}$ along the decomposition regions specified by the lead. Information collected by the motion planner such as coverage and time is fed back to the discrete-search component to improve the lead computed in the next iteration. Fig. 2 illustrates the interplay of discrete-search and motion-planning components of `HyDICE` and Algorithm 2 provides pseudocode for `HyDICE`. The discrete-search and motion-planning components correspond to line 6 and lines 7–14 of Algorithm 2 and are described in sections 3.2 and 3.3, respectively.

3.2 Discrete-Search Component of `HyDICE`

3.2.1 Decomposition

The decomposition D of the continuous state spaces associated with the modes of the hybrid system (Algorithm 2:3) is obtained by decomposing each $X_q \in X$ into a number of different regions, i.e., $D = \{D(q) : q \in Q\}$ and $D(q) = \{D_1(q), \ldots, D_{n_q}(q)\}$. `HyDICE` does not impose any strict requirements on the decomposition and each $D(q)$ is usually computed as a set of nonoverlapping regions in some low-dimensional projection of X_q. The objective of the projection is to reduce the dimensionality, while at the same time

Algorithm 2 Pseudocode for `HyDICE`

```
    Input: $H = (S, Inv, E, G, J, f, U, I, F)$: hybrid system
           $t_{\max} \in \mathbb{R}$: upper bound on overall computation time
           $t_\sigma \in \mathbb{R}$: short time allocated to each motion planning step
    Output: A witness trajectory or FAILURE if no witness trajectory is found

 1: StartClock
 2: $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$; $V_\mathcal{T} \leftarrow \{s_{\text{safe}}\}$; $E_\mathcal{T} \leftarrow \emptyset$     ◇initialize search tree
 3: $D \leftarrow$ Decomposition($H$)
 4: $G_D = (V_D, E_D) \leftarrow$ DiscreteSearchGraph($H, D$)
 5: while ElapsedTime $< t_{\max}$ do
 6:     $\sigma \leftarrow$ DiscreteSearch($G_D$)     ◇compute current lead $\sigma$
 7:     StartClock2     ◇extend $\mathcal{T}$ along regions specified by $\sigma$
 8:     while ElapsedTime2 $< t_\sigma$ do     ◇begin motion-planning step
 9:     $D_i(q) \leftarrow$ SelectDecompositionRegion($\mathcal{T}, \sigma$)
10:     $s \leftarrow$ SelectStateFromDecompositionRegion($D_i(q)$)
11:     $s_{\text{new}} \leftarrow$ PropagateForward($H, \mathcal{T}, s, \sigma$)
12:     $V_\mathcal{T} \leftarrow V_\mathcal{T} \cup \{s_{\text{new}}\}$; $E_\mathcal{T} \leftarrow E_\mathcal{T} \cup \{(s, s_{\text{new}})\}$
13:     if $s_{\text{new}} \in F$ then
14:         return WitnessTrajectory($\mathcal{T}, s_{\text{new}}$)     ◇end motion-planning step
15: return FAILURE
```

preserve the underlying structure of the original set. As such, the projection is state-space dependent, and it is generally suggested by the user. For many systems, simple projections that consider only some of the state components have been shown to work well in practice [34, 45].

For the hybrid system used in this work, `HyDICE` projects each X_q onto $\mathbb{R}^2$ and constructs a grid decomposition with $n_r(q)$ rows and $n_c(q)$ columns. More specifically, let $\text{proj}_q : X_q \to \mathbb{R}^2$ compute the projection of each $x \in X_q$ onto $[a_{\min}(q), a_{\max}(q)] \times [b_{\min}(q), b_{\max}(q)] \subset \mathbb{R}^2$. Then, $n_q = n_c(q) \times n_r(q)$, and for each $i = \{1, \dots, n_q\}$,

$$D_i(q) = \{x \in X_q : \text{proj}_q(x) \in [a_i(q), a_i(q) + \alpha(q)) \times [b_i(q), b_i(q) + \beta(q))\},$$

where $a_i(q) = a_{\min}(q) + c\alpha(q)$; $b_i(q) = b_{\min}(q) + r\beta(q)$; $c = (i-1) \mod n_c(q)$; $r = (i-1) \div n_c(q)$; $\alpha(q) = (a_{\max}(q) - a_{\min}(q))/n_c(q)$; and $\beta(q) = (b_{\max}(q) - b_{\min}(q))/n_r(q)$. Other types of decompositions are also possible and are discussed in Section 5.

3.2.2 Discrete-search graph

`HyDICE` uses the decomposition D and the discrete transitions of the hybrid system to construct a search graph $G_D = (V_D, E_D)$, as illustrated in Algorithm 2:4. A vertex $v_i(q)$ is added to V_D for each $D_i(q)$. In addition, V_D contains two special vertices v_{safe} and v_{unsafe}. An edge $(v_{\text{safe}}, v_i(q))$ is added to E_D for every $D_i(q)$ such that $D_i(q) \cap I \neq \emptyset$. Similarly, an edge $(v_i(q), v_{\text{unsafe}})$ is added to E_D for every $D_i(q)$ such that $D_i(q) \cap F \neq \emptyset$. Furthermore, an edge $(v_i(q), v_j(q))$ is added to E_D if the projections of $D_i(q)$ and $D_j(q)$ are adjacent, i.e., $||(a_i(q), b_i(q)), (a_j(q), b_j(q))||_2 \leq \sqrt{\alpha^2(q) + \beta^2(q)}$. Finally, an edge $(v_i(q'), v_j(q''))$ is added to E_D if there is a discrete transition from some state (q', x'), $x' \in D_i(q')$, to some state (q'', x''), $x'' \in D_j(q'')$. `HyDICE` uses a function RegionTrans($D_i(q'), D_j(q'')$) to determine if there is the possibility of a discrete transition from $D_i(q')$ to $D_j(q'')$.

Note that the computation of RegionTrans($D_i(q'), D_j(q'')$) depends on the definition of the guard $G_{(q',q'')}$ and reset function $J_{(q',q'')} : G_{(q',q'')} \to X_{q''}$. When it is computationally infeasible or expensive to determine if there is a discrete transition from $D_i(q')$ to $D_j(q'')$, the definition of $\textsc{RegionTrans}(D_i(q'), D_j(q''))$ can be relaxed. In fact, it is only required that $\textsc{RegionTrans}(D_i(q'), D_j(q''))$ does not return any false negatives, i.e., $\textsc{RegionTrans}(D_i(q'), D_j(q''))$ returns false when there is a discrete transition from $D_i(q')$ to $D_j(q'')$. A false negative would cause `HyDICE` to miss an edge in the search graph $G_D = (V_D, E_D)$, and as a result, not be able to compute any feasible leads. False positives, i.e., $\textsc{RegionTrans}(D_i(q'), D_j(q''))$ returns true when there is in no discrete transition from $D_i(q')$ to $D_j(q'')$, are however allowed. A false positive would add a spurious edge to the search graph $G_D = (V_D, E_D)$, which could lead to the computation of an infeasible lead. However, as the search progresses, the weight estimates associated with the spurious edge would indicate that such edge should not be included in future leads as it is not helping `HyDICE` to extend the search tree $\mathcal{T}$. By allowing false positives, the computation of $\textsc{RegionTrans}(D_i(q'), D_j(q''))$ can be greatly simplified. In particular, it can be computed in any of the following ways:

– $\textsc{RegionTrans}(D_i(q'), D_j(q'')) = \top \iff D_i(q') \cap G_{(q',q'')} \neq \emptyset$
– $\textsc{RegionTrans}(D_i(q'), D_j(q'')) = \top \iff (q', q'') \in E$

3.2.3 Computation of leads

The current lead σ is computed at each iteration (Algorithm 2:6) by searching the graph $G_D = (V_D, E_D)$ for a sequence of edges that connects v_{safe} to v_{unsafe}. A central issue is which lead σ to select from the set Γ of all possible leads. Assume for the moment that $w(\sigma) > 0$ reflects a running estimate on the likelihood σ is useful to `HyDICE` for constructing a witness trajectory. An effective strategy that balances greedy search with methodical search can be obtained by selecting each lead σ with probability $w(\sigma)/\sum_{\sigma' \in \Gamma} w(\sigma')$. This selection process is biased towards the most useful leads, since the objective of `HyDICE` is to quickly construct a witness trajectory. At the same time, since it is not known a priori which σ actually leads to the construction of a witness trajectory, the selection process guarantees that each $\sigma \in \Gamma$ has a non-zero probability of being selected. Computationally however such selection strategy is feasible only when it is practical to enumerate all leads. Due to the decomposition and the potentially huge complexity of discrete transitions, there is usually a combinatorial number of leads, which makes enumeration impractical.

The approach followed in this work addresses this issue by maintaining instead a running estimate $w_i(q)$ on the priority of including the decomposition region $D_i(q)$ in the current lead σ. The weight is computed as

$$w_i(q) = \frac{\text{vol}^{\nu}(D_i(q))\text{cov}^{\zeta}(D_i(q))}{t^{\tau}(D_i(q))},$$

where $t(D_i(q))$ is the time the motion planner has spent extending the search tree $\mathcal{T}$ from states associated with $D_i(q)$, i.e., time spent by $\textsc{PropagateForward}$ in Algorithm 2:11; $\text{vol}(D_i(q))$ is the volume of the projection of $D_i(q)$, i.e., $\text{vol}(D_i(q)) = \alpha(q)\beta(q)$; $\text{cov}(D_i(q))$ measures the coverage of $D_i(q)$ by $\mathcal{T}$, which is computed by imposing an implicit fine-grained uniform grid on the projection of $D_i(q)$ and measuring the number of cells that contain at least one state from $\mathcal{T}$; and τ, ν, ζ are normalization constants. Note that a new cell c is added to the implicit uniform grid only when a state $s_{\text{new}} = (q, x) \in S$ is added to $V_{\mathcal{T}}$ such that $\text{proj}_q(x) \in c$. A hash-set data structure is used by HyDICE to keep track of which cells have currently been added to the implicit uniform grid and update the coverage estimate in roughly constant time.

A high weight $w_i(q)$ indicates priority. When the coverage estimate $\text{cov}(D_i(q))$ of a decomposition region $D_i(q)$ is high, then there are many vertices and edges which HyDICE can use to extend $\mathcal{T}$ from $D_i(q)$ to the next decomposition region in the lead. Preference is also given to $D_i(q)$ when it has a large volume, since it allows HyDICE to extend $\mathcal{T}$ in different directions. The time estimation factor $t(D_i(q))$ ensures that HyDICE does not spend all the computation time in one particular decomposition region. In fact, as $t(D_i(q))$ increases, the likelihood that $D_i(q)$ is included in the current lead decreases rapidly, allowing HyDICE to spend time searching inside other decomposition regions. The weighting function $w_i(q)$ is thus biased towards decomposition regions that have large volume and are quickly covered by the search tree.

The current lead σ is then obtained as the shortest path from v_{safe} to v_{unsafe} in the graph $G_D = (V_D, E_D)$, where the path length is determined by the weights $w(v_i(q'), v_j(q'')) = 1/(w(D_i(q')) * w(D_j(q'')))$ associated with each edge $(v_i(q'), v_j(q'')) \in E_D$. The shortest path can be efficiently computed using A* or Dijkstra's algorithm. For considerably larger problems, more advanced graph-search techniques [56] or approaches from model checking, such as bounded model checking [13] or directed model checking [17], could be used (see also discussion in Section 5). The current lead σ, with a small probability, is also computed as a random path from v_{safe} to v_{unsafe} as a way to correct for errors inherent with the weight estimates and to ensure that each lead has a non-zero probability of being selected. In this way, the discrete-search component is able to lead the search for a witness trajectory toward promising directions, while allowing the motion planner to extend the search tree along new directions.

3.3 Motion-Planning Component of HyDICE

The objective of the motion planner is to extend the search tree $\mathcal{T}$ along the decomposition regions associated with the current lead σ so that $\mathcal{T}$ can reach F as quickly as possible. This is achieved by selecting states from the decomposition regions specified by σ and propagating forward from those states.

Conceptually, forward propagation provides the necessary mechanism for the motion planner to extend $\mathcal{T}$ and search the state space of the hybrid system. The forward propagation from a state $s = (q, x) \in S$ entails applying a control u to s and simulating the continuous and discrete dynamics of the hybrid system for a certain duration of time T to obtain a new state $s_{\text{new}} \in S$. The state s_{new} thus corresponds to the last state of the trajectory $\Upsilon_{s,u,T}$, as described in Definition 2. The control $u \in U_q$ is usually selected pseudo-uniformly at random from the set of all possible controls or according to some specific control law that selects controls depending on state values and other criteria, as illustrated in Section 4. The new state s_{new} and the edge (s, s_{new}) are added to the vertices and edges of $\mathcal{T}$, respectively.

As indicated in Algorithm 2:7–14, the motion planner repeats the above select-and-propagate step until an upper bound t_σ on the time dedicated to σ is exceeded. It is important that the motion planner commits to the current lead σ only for a short period of time t_σ to allow for an effective interplay with the discrete-search component, since leads are continually refined based on information collected during the search and potentially new leads are computed at the beginning of each iteration step (Algorithm 2:6). The rest of this section describes in more detail the selection of a decomposition region $D_i(q)$ from the decomposition regions associated with σ (Algorithm 2:9), selection of a state $s \in V_{\mathcal{T}}$ from the states associated with $D_i(q)$ (Algorithm 2:10), and the forward propagation from s to a new state s_{new}.

3.3.1 Selection of a Decomposition Region

Since the objective of the motion planner is to extend $\mathcal{T}$ toward F, the function SELECTDECOMPOSITIONREGION (Algorithm 2:9) gives preferences to those decomposition regions of σ that have been reached by $\mathcal{T}$ and are closer to F. Note that $D_i(q)$ is reached by $\mathcal{T}$ when a state $s = (q, x) \in V_{\mathcal{T}}$ and $x \in D_i(q)$. Furthermore, the order in which $v_i(q)$ appears in σ is an indication of how close $D_i(q)$ is to F.

More specifically, the motion planner maintains a set D_{avail} of decomposition regions that are available for the selection process. Initially, $D_{\text{avail}} = \emptyset$. The lead σ is scanned backwards starting at position $i = |\sigma|$ down to $i = 1$. If the i-th decomposition region $D_i(q)$ of σ is reached by $\mathcal{T}$, then $D_i(q)$ is added to D_{avail} with probability $1/(1 + |D_{\text{avail}}|)$. Thus, decomposition regions that have been reached by $\mathcal{T}$ and appear toward the end of σ are estimated to be closer to F and are thus given a higher priority by the motion planner. Each $D_i(q) \in D_{\text{avail}}$ is then selected with probability

$$\frac{w_{\text{sel}}(D_i(q))}{\sum_{D_j(q') \in D_{\text{avail}}} w_{\text{sel}}(D_j(q'))},$$

where

$$w_{\text{sel}}(D_i(q)) = \frac{\text{vol}^{\nu}(D_i(q))}{t^{\tau}(D_i(q))\text{cov}^{\zeta}(D_i(q))}.$$

This selection strategy allows the motion planner to spend more time extending the search tree $\mathcal{T}$ along those decomposition regions that are close to F, have large volume, and have not been adequately covered in the past.

When $s_{\text{new}} = (q_{\text{new}}, x_{\text{new}}) \in S$ is added to $\mathcal{T}$ (Algorithm 2:12), s_{new} is also added to the appropriate decomposition region $D_j(q_{\text{new}})$, such that $x_{\text{new}} \in D_j(q_{\text{new}})$. If $D_j(q_{\text{new}})$ is not already in D_{avail}, then $D_j(q_{\text{new}})$ is added to D_{avail}. Thus, when the motion planner extends $\mathcal{T}$ along new decomposition regions, they become available for selection during the next iteration of the motion-planning step in Algorithm 2:9. In this way, the motion planner extends $\mathcal{T}$ along decomposition regions associated with σ and along new decomposition regions that $\mathcal{T}$ reaches while the search for a witness trajectory progresses from one decomposition region to another.

3.3.2 Selection of a State from a Decomposition Region

As illustrated in Algorithm 2:10, among all the states in $V_{\mathcal{T}}$ associated with $D_i(q)$, the function SELECTSTATEFROMDECOMPOSITIONREGION selects one state s from which it extends $\mathcal{T}$. The state-selection strategy follows well-established techniques developed in motion planning research, similar to the work in [50]. Recall that an implicit uniform grid was used to estimate the coverage of $D_i(q)$ by the states in $V_{\mathcal{T}}$, as discussed in Section 3.2. The i-th cell from this implicit uniform grid is selected with probability $(1/\text{nsel}_i^2)/\sum_j(1/\text{nsel}_j^2)$, where nsel_i is the number of times the i-th cell has been selected in the past. A state s is then selected pseudo-uniformly at random from all the states associated with the i-th cell. This state-selection strategy gives priority to new states that have not been frequently selected in the past and allows the motion planner to extend $\mathcal{T}$ along new directions.

3.3.3 Extending the Tree from the Selected State by Forward Propagation

As mentioned earlier, the actual extension of $\mathcal{T}$ from $s = (q, x)$ is computed by the PropagateForward function in Algorithm 2:11. An input control $u \in U_q$, which could be selected pseudo-uniformly at random or according to some other strategy (see Section 4.1.2), is applied to s for a short duration of time $T > 0$. The function PropagateForward simulates the continuous and discrete dynamics of the hybrid system to obtain the resulting hybrid-system trajectory $\Upsilon_{s,u,T}$, as in Definition 2. Pseudocode is given in Algorithm 3.

Algorithm 3 PropagateForward

```
    Input: $H = (S, Inv, E, G, J, f, U, I, F)$: hybrid system
           $s = (q, x) \in S$: starting state
           $\epsilon \in \mathbb{R}^{>0}$: integration step
           $n_{\text{steps}} \in \mathbb{N}$: number of integration steps
    Output: The new state obtained at the end of propagation
 1: $u \leftarrow$ sample control from $U_q$
 2: $x_0 \leftarrow x$
 3: for $i = 1, 2, \ldots, n_{\text{steps}}$ do
 4:     $x_i \leftarrow \int_0^{\epsilon} f_q(x_{i-1}, u)$
 5:     if $x_i \notin Inv_q$ then
 6:         return $(q, x_{i-1})$
 7:     if $(q, x_i) \in G_{(q,q_{\text{new}})}$ for some $q_{\text{new}} \in Q$ then
 8:         $(x_{\text{loc}}, T) \leftarrow$ localize discrete event in time interval $((i-1) * \epsilon, i * \epsilon]$
 9:         return $J_{(q,q_{\text{new}})}(q, x_{\text{loc}})$
10: return $(q, x_i)$
```

The forward propagation follows the continuous dynamics f_q associated with $q \in Q$ and is usually computed based on numerical integration of the ordinary differential equations associated with f_q. This work uses 8-th order Prince-Dormand Runge-Kutta numerical integration with adaptive step-size control as implemented in GSL [21]. The forward propagation is an iterative procedure. Let n_{steps} denote the number of propagation steps and let $\epsilon > 0$ denote the integration step. Initially, $x_0 = x$ (Algorithm 3:2). During the i-th iteration, the continuous state $x_i \in X_q$ is obtained by numerically integrating the differential equations $f_q(x_{i-1}, u)$ for ϵ units of time (Algorithm 3:4).

If $x_i \notin Inv_q$, then the forward propagation is terminated, since x_i is not valid (Algorithm 3:5–6). The previous valid state (q, x_{i-1}) is returned as the new state s_{new} obtained at the end of the forward propagation. The valid hybrid-system trajectory corresponds then to $\Psi_{s,u,T}$ (see Definition 2), where $T = (i-1) * \epsilon$.

If $x_i \in Inv_q$, the simulation checks whether the state (q, x_i) satisfies any guard condition, i.e., $(q, x_i) \in G_{(q,q_{\text{new}})}$ for some $q_{\text{new}} \in Q$. If a guard condition is satisfied, then a discrete event has occurred in the time interval $(i - 1 * \epsilon, i * \epsilon]$ (Algorithm 3:7). This stage, commonly known as *event detection*, is followed by the *event localization* stage, which localizes the earliest time $T \in ((i-1)*\epsilon, i*\epsilon]$ the guard condition is satisfied (Algorithm 3:8). Variants of bisection or bracketing algorithms, as those found in the classical numerical literature, are commonly employed for the event detection [18]. Once the event is localized, the propagation stops and the corresponding discrete transition is applied to obtain the new state s_{new} (Algorithm 3:9). The valid hybrid-system trajectory corresponds then to $\Upsilon_{s,u,T}$ (see Definition 2).

At the end of the forward propagation, the new state $s_{\mathrm{new}} = \Upsilon_{s,u,T}(T)$ and the edge (s, s_{new}) are added to the vertices and edges of $\mathcal{T}$, respectively (Algorithm 2:12). A witness trajectory is found if $s_{\mathrm{new}} \in F$. The witness trajectory is computed by reconstructing the evolution of the hybrid system from s_{safe} to s_{new} following the appropriate edges of $\mathcal{T}$ (Algorithm 2:14).

We note that, due to limitations of floating-point arithmetic, as with any other numerical method in computational mathematics, including symbolic techniques for linear hybrid systems, there will be round-off errors in the simulation of the continuous dynamics and the event detection and localization of discrete transitions. The approach followed by `HyDICE` to deal with such numerical errors is similar to the approach followed by other numerical methods for hybrid-system falsification [7,32,41], which choose the integration step $\epsilon > 0$ to be small in order to minimize such errors. For certain hybrid systems with linear guard descriptions, it is also possible to use more accurate event detection and localization algorithms, such as those surveyed and developed in [18], which come asymptotically close to the boundary of the guard set without overshooting it.

4 Experiments and Results

Experimental validation is provided by using `HyDICE` for the falsification of safety properties of a hybrid robotic system navigation benchmark and an aircraft conflict-resolution protocol. The navigation benchmark, which is based on a scalable benchmark proposed in [20], tests the scalability of `HyDICE` with respect to the number of modes. The aircraft conflict-resolution protocol, which has been widely used in [7, 32, 41, 53], tests the computational efficiency of `HyDICE` when also dealing with high-dimensional continuous states spaces.

Methods used for the Comparisons An important part of experiments is the comparison with previous related work. The closest work we can compare to is the application of `RRT` to hybrid systems [19, 32]. We also compare our work to a more recent version of `RRT` developed in [41] as a hybrid-system falsification method that is guided by the star discrepancy coverage measure. To distinguish between `RRT` and its variant, we will use the acronym `RRT[D`*`]` to refer to the star-discrepancy version of `RRT` [41]. We also provide experiments that indicate the impact of the discrete-search component on the computational efficiency of `HyDICE`. We refer to the version of `HyDICE` that does not use the discrete-search component as `HyDICE[NoLeads]`. From an algorithmic perspective, `HyDICE[NoLeads]` is the sampling-based motion planner of `HyDICE`. More precisely, `HyDICE[NoLeads]` is obtained from Algorithm 2 by commenting out the outer while loop in line 5 and setting $t_\sigma = t_{\max}$.

Hardware Experiments were run on the Rice Cray XD1 ADA and PBC clusters, where each processor is at 2.2GHz and has up to 8GB RAM. Each run uses a single processor, i.e., no parallelism. An upper bound of 3600s is set for each run. In the case of `HyDICE`, the current lead σ is computed as the shortest path in the search graph with probability 0.9 and as a random path with probability 0.1 (see Section 3).

4.1 A Hybrid Robotic System Navigation Benchmark

The first hybrid-system benchmark used in the experiments consists of an autonomous robotic vehicle, whose underlying dynamics change discretely depending on terrain conditions. The choice of this specific system is to provide a concrete, scalable benchmark in which the competitiveness of `HyDICE` can be tested. This hybrid-system benchmark, which is motivated by robotics applications, is constructed based on a scalable navigation benchmark proposed in [20]. A given environment is divided into $n \times n$ equally sized terrains. The hybrid robotic system associates one mode $q_i \in Q$ with each terrain R_i. For each mode, the associated dynamics is specified by a set of ordinary differential equations, as described in Section 4.1.1. A discrete transition $(q_i, q_j) \in E$ occurs when the hybrid robotic system moves from R_i to R_j. When the discrete transition occurs, velocity components of the current continuous state of the hybrid robotic vehicle are set to zero.

4.1.1 Second-order models

While the navigation benchmark proposed in [20] used linear dynamics, this work uses second-order dynamics that are commonly used for modeling cars, differential drives, and unicycles. Detailed descriptions of these models can be found in [11, 37].

Smooth car (`SCar`): A second-order car is controlled by setting the acceleration and the rotational velocity of the steering wheel. The dynamics is specified as $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = v\tan(\phi)/L$; $\dot{v} = u_0$; $\dot{\phi} = u_1$, where (x, y, θ) is the configuration; $L = 0.8m$ is the distance between the front and rear axles; $|v| \leq v_{\max} = 3m/s$ is the velocity; $|\phi| \leq \phi_{\max} = 40^\circ$ is the steering angle; $|u_0| \leq \max_0 = 0.8m/s^2$ is the acceleration control; and $|u_1| \leq \max_1 = 25^\circ/s$ is the control for the steering wheel.

Smooth unicycle (`SUni`): A second-order unicycle is controlled by translational and rotational accelerations. The dynamics is given by $\dot{x} = v\cos(\theta)$; $\dot{y} = v\sin(\theta)$; $\dot{\theta} = \omega$; $\dot{v} = u_0$; $\dot{\omega} = u_1$, where (x, y, θ) is the configuration; $|v| \leq v_{\max} = 3m/s$ and $|\omega| \leq \omega_{\max} = 20^\circ/s$ are the translational and rotational velocities; $|u_0| \leq \max_0 = 0.3m/s^2$ and $|u_1| \leq \max_1 = 10^\circ/s^2$ are the translational and rotational acceleration controls.

Smooth differential drive (`SDDrive`): A second-order differential drive is controlled by setting the left and right wheel rotational accelerations. The dynamics is given by $\dot{x} = 0.5r(\omega_\ell + \omega_r)\cos(\theta)$; $\dot{y} = 0.5r(\omega_\ell + \omega_r)\sin(\theta)$; $\dot{\theta} = r(\omega_r - \omega_\ell)/L$; $\dot{\omega_\ell} = u_0$; $\dot{\omega_r} = u_1$, where (x, y, θ) is the configuration; $|\omega_\ell| \leq \omega_{\max} = 5^\circ/s$ and $|\omega_r| \leq \omega_{\max}$ are the rotational velocities of the left and right wheels; $r = 0.2m$ is the wheel radius; $L = 0.8m$ is the length of the axis connecting the centers of the two wheels; $|u_0| \leq \max_0 = 10^\circ/s^2$ and $|u_1| \leq \max_1 = 10^\circ/s^2$ are the left and right wheel acceleration controls.

4.1.2 Autonomous driver models

The controls u_0 and u_1 could be thought of as playing the role of the automatic driver. The objective of hybrid-system falsification is then to test the safety of the automatic driver, i.e., the driver is unsafe if a witness trajectory is produced that indicates that it is possible for the robotic vehicle to enter an unsafe state. The driver models used in this work consist of simple if-then-else statements that depend on the state values and the underlying dynamics associated with each mode of the hybrid robotic system.

In the first model, `RandomDriver`, u_0 and u_1 are selected pseudo-uniformly at random from $[-\max_0, \max_0]$ and $[-\max_1, \max_1]$, respectively. In the second model, `StudentDriver`, the driver follows an approach similar to stop-and-go. When the speed is close to zero, `StudentDriver` selects u_0 and u_1 as in `RandomDriver`. Otherwise, `StudentDriver` selects controls that reduce the speed. The third model, `HighwayDriver` attempts to maintain the speed within acceptable low and upper bounds. When the speed is too low, `HighwayDriver` selects controls that increase the speed. When the speed is too high, `HighwayDriver` selects controls that slow down the robotic vehicle. Otherwise, `HighwayDriver` selects controls that do not change the speed considerably. For completeness, we provide below a succinct description of the selection strategy for the input controls u_0 and u_1 for each driver model and each second-order dynamics:

```
RandomDriver $f(a, i, c, L, R)$: return $\mathrm{rnd}(-\max_i, \max_i)$
StudentDriver $f(a, i, c, L, R)$:
  if $a \in (0.2, 1]$ then return $\mathrm{rnd}(-L\max_i, R(c-1)\max_i)$
  elif $a \in [-1, -0.2)$ then return $\mathrm{rnd}(R(1-c)\max_i, L\max_i)$
  else return $\mathrm{rnd}(-\max_i, \max_i)$
HighwayDriver $f(a, i, c, L, R)$: B={0.4, 0.6, 0.8, 1.0}
  if $\exists b \in B \wedge a \in (b-0.2, b]$ then return $\mathrm{rnd}(-Lb\max_i, R(c-b)\max_i)$
  elif $\exists b \in B \wedge a \in [-b, 0.2-b)$ then return $\mathrm{rnd}(R(b-c)\max_i, Lb\max_i)$
  else return $\mathrm{rnd}(-\max_i, \max_i)$
SCar: $u_0 = f(v/v_{\max}, 0, 0.2, 1, 1)$; $u_1 = \mathrm{rnd}(-\max_1, \max_1)$
SUni: $u_0 = f(v/v_{\max}, 0, 0.2, 1, 1)$; $u_1 = f(\omega/\omega_{\max}, 1, 0.2, 1, 1)$
SDDrive: $u_0 = f(a, 0, 1.2, 0, 0.25)$; $u_1 = -f(a, 1, 1.2, 0, 0.25)$; $a = \frac{\omega_\ell + \omega_r}{2\omega_{\max}}$
```

4.1.3 Modes and discrete transitions

The continuous dynamics associated with each mode $q \in Q$ is selected pseudo-uniformly at random from `SCar`, `SUni`, and `SDDrive`. The set of discrete transitions E is created using a strategy similar to maze generation based on Kruskal's algorithm [33]. Initially, E is empty and walls are placed between each pair of neighboring terrains R_i and R_j. Then, walls are visited in some random order. If the terrains divided by the current wall belong to distinct sets, then the wall is removed and the two sets are joined. At the end, each remaining wall is kept with probability $p = 0.9$ to allow for more than one passage from one terrain to another. Each time a wall that separates some terrain R_i from R_j is removed, discrete transitions (q_i, q_j) and (q_j, q_i) are added to E.

4.1.4 Experiments

Experiments are performed using the hybrid robotic system described in Section 4.1. A problem instance is obtained by fixing the number of modes $|Q| = n \times n$ and the driver model to `RandomDriver`, `StudentDriver`, or `HighwayDriver`. For each problem instance, we create 40 safety properties. Each safety property is created by selecting pseudo-uniformly at random one terrain as the initial place where the search for a witness trajectory should start and another terrain as unsafe. A violation of the safety property then occurs when the hybrid robotic vehicle enters the unsafe terrain. For

RandomDriver	$\lvert Q\rvert$									
	1^2	2^2	4^2	8^2	16^2	32^2	64^2	128^2	512^2	1024^2
RRT	0.1	0.1	0.3	1.5	16.8	195.3	X	X	X	X
RRT[D*]	0.1	0.9	0.5	4.7	5.1	24.8	411.3	X	X	X
HyDICE[NoLeads]	0.1	0.1	0.3	3.6	5.7	10.0	147.1	564.8	X	X
HyDICE	0.4	0.4	0.6	1.2	1.5	2.4	11.1	66.1	352.7	1198.4

StudentDriver	$\lvert Q\rvert$									
	1^2	2^2	4^2	8^2	16^2	32^2	64^2	128^2	512^2	1024^2
RRT	0.1	0.2	0.7	2.4	25.4	210.5	X	X	X	X
RRT[D*]	0.1	1.4	0.3	1.0	4.6	23.2	605.8	X	X	X
HyDICE[NoLeads]	0.1	0.1	0.4	3.4	5.6	10.3	189.2	576.8	X	X
HyDICE	0.4	0.4	0.7	1.3	1.8	3.4	12.4	64.6	294.5	1289.9

HighwayDriver	$\lvert Q\rvert$									
	1^2	2^2	4^2	8^2	16^2	32^2	64^2	128^2	512^2	1024^2
RRT	0.1	0.2	0.3	2.9	25.5	219.3	X	X	X	X
RRT[D*]	0.2	0.7	0.2	0.9	3.9	23.7	515.5	X	X	X
HyDICE[NoLeads]	0.1	0.1	0.4	4.0	5.9	8.8	151.6	514.9	X	X
HyDICE	0.4	0.4	0.6	1.3	1.8	2.9	10.9	70.4	288.9	954.8

Table 1 Computational efficiency of HyDICE compared to other methods as a function of the number of modes $|Q|$ and the driver model. Times are in seconds. Entries marked with X indicate a timeout, which was set to 3600s.

each experiment, we report the average computational time in seconds. Recall that an upper bound of 3600s was set for each run.

4.1.5 Results

A summary of the results is shown in Table 1. Table 1 indicates that HyDICE is consistently more efficient than RRT. As an example, when RandomDriver is used and $|Q| = 32^2$, RRT requires on average 195.3s, while HyDICE requires only 2.4s. Similarly, when StudentDriver or HighwayDriver are used and $|Q| = 32^2$, RRT requires on average 210.5s and 219.3s, while HyDICE requires only 3.4s and 2.9s, respectively. Moreover, as the number of modes is increased HyDICE remains efficient, while RRT times out. As Table 1 shows, RRT times out in all instances with $|Q| \geq 64^2$, while HyDICE requires on average less than 15s for problem instances with $|Q| = 64^2$ and less than 75s for problem instances with $|Q| = 128^2$.

Table 1 also indicates that HyDICE is consistently more efficient than RRT[D*]. The computational advantages of HyDICE become more pronounced as $|Q|$ is increased. For example, when RandomDriver is used and $|Q| = 64^2$, RRT[D*] requires on average 411.3s. Similarly, when StudentDriver or HighwayDriver are used, RRT[D*] requires 605.8s and 515.5s, respectively. On the other hand, as mentioned earlier, HyDICE requires on average less than 15s. Furthermore, RRT[D*] times out as the number of modes is increased to $|Q| = 128^2$, while HyDICE requires only a short time (less than 75s) to handle such problem instances.

The second set of experiments provides insight on the observed computational efficiency of HyDICE. In particular, we investigate the importance of the discrete-search component on HyDICE. As noted earlier, HyDICE[NoLeads] is the version of HyDICE that does not use leads to guide the motion planner during the search for a witness trajectory. Table 1 shows that although `HyDICE[NoLeads]` is still faster than `RRT` and `RRT[D*]`, it is considerably slower than `HyDICE`. (For a discussion on issues related to the computational efficiency of `RRT` and sampling-based motion planners similar to `HyDICE[NoLeads]` see [11,37,44,45].) For example, `HyDICE[NoLeads]` is capable of handling problem instances even with $|Q| = 128^2$, while both `RRT` and `RRT[D*]` timed out on these problem instances. However, `HyDICE[NoLeads]` requires on the average 564.8s, 576.8s, and 514.9s when `RandomDriver`, `StudentDriver`, and `HighwayDriver` are used, respectively, while `HyDICE` requires only 66.1s, 64.6s, and 70.4s. These results highlight the importance of the discrete-search component, which, by guiding the motion planner during the search for a witness trajectory, significantly improves the computational efficiency of `HyDICE`.

Table 1 also shows that `HyDICE` scales up reasonably well and can handle nonlinear problem instances with over a million modes. While other methods failed to handle large problem instances beyond $|Q| = 128^2$, `HyDICE` even when $|Q| = 1024^2$ remains computationally efficient. Overall, results in Table 1 show the competitiveness of `HyDICE` as a hybrid-system falsification method.

4.2 Aircraft Conflict-Resolution Protocol

The aircraft conflict-resolution protocol, which has been widely used in [7,32,41,53], tests the computational efficiency of `HyDICE` when also dealing with high-dimensional continuous states spaces.

The continuous state space is $X = X_1 \times X_2 \times \cdots \times X_N$, where X_i is the continuous state space associated with the i-th aircraft. Each aircraft i has three continuous state variables (x_i, y_i, θ_i), where (x_i, y_i) denotes the position and θ_i denotes the orientation.

This work presents experiments with up to 20 aircraft (60 continuous dimensions), which is considerably larger than instances considered in related work (5 aircraft in [32] and 8 aircraft in [41]). The continuous dynamics of the i-th aircraft are given by

$$\begin{aligned}
\dot{x}_i &= v\cos(\theta_i) + (-u_1\sin(\theta_i) + d_2\cos(\theta_i))(-\sin(\theta_i)) \\
\dot{y}_i &= v\cos(\theta_i) + (-u_1\sin(\theta_i) + d_2\cos(\theta_i))(\cos(\theta_i)) \\
\dot{\theta}_i &= \textsc{Protocol}(i)
\end{aligned}$$

where v is a constant forward velocity; $u_1, u_2 \in [-w, w]$ is the wind disturbance; and $\textsc{Protocol}(i)$ determines the yaw rate. The discrete dynamics, which makes this benchmark a hybrid system, are incorporated in the computation of $\textsc{Protocol}(i)$, which is based on a conflict-resolution protocol that aims to safely bring all aircrafts from their initial positions $(x_i^{\text{init}}, y_i^{\text{init}})$ to their goal positions $(x_i^{\text{goal}}, y_i^{\text{goal}})$ while avoiding collisions with each other.

As in [19, 32, 41], the function $\textsc{Protocol}(i)$ switches depending on the modes associated with the aircrafts. At the initial position, the i-th aircraft is in heading mode, $q = 1$, and rotates with an angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = \theta_{\text{goal}} - \theta_i$ until it points toward the goal position, where $\theta_{\text{goal}} \in [-\pi, \pi)$ is computed as the directed angle between the x-axis and $(x_i^{\text{goal}}, y_i^{\text{goal}})$. Once reaching the desired goal heading, the i-th aircraft switches to cruising mode, $q = 2$, and cruises toward the goal with angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = 0$. If two aircrafts i and j get close to each-other, i.e., within p distance, then both aircrafts enter an avoid mode, $q = 2$. During the avoid mode, both aircrafts i and j make an instantaneous turn by -90°

and then follow a half-circle with constant angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = c$ and $\dot{\theta}_j = \textsc{Protocol}(j) = c$. At the end of the half circle, each aircraft makes instantaneous turns until pointing toward their own goal positions, and then the aircrafts return to cruise mode. It is also possible that during the avoid mode between aircrafts i and j, another aircraft k comes within p distance to i. In this case, aircrafts i and k make instantaneous turn by -90° and execute the same avoid procedure as above. When an aircraft reaches the goal position, it stays there and it is no longer involved in the collision-avoidance protocol.

A violation of the safety property occurs if at any point two aircraft come within d ($d < p$) distance from each other.

We initially experimented with the benchmark in [19], which has 5 aircraft (15 continuous dimensions). As in [19], the avoidance distance was set to $p = 5.25km$ and the collision distance was set to $d = 1km$. The translational velocity was set to $0.3km/s$ and the angular velocity was set to $c = 0.03rad/s$. The maximum wind disturbance was set to $w = 0.1$. For the benchmark used in [19], all computational methods tested in the experiments, `RRT`, `RRT[D*]`, `HyDICE[NoLeads]`, `HyDICE`, were able to compute witness trajectories in a matter of a few seconds (less than 10s). The methods would quickly find collisions that resulted from the aircrafts making instantaneous -90° turns during the avoid mode and bumping into each other as they followed the respective half-circles. Fig. 3 provides an illustration.

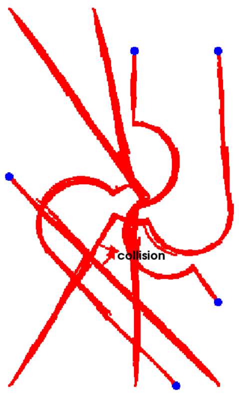

Fig. 3 A collision between two aircraft is quickly found after a few seconds (less than 10s) of exploration. The exploration is shown in red. Goal positions are shown as blue circles.

4.2.1 Safer Aircraft Conflict-Resolution Protocol

In order to make the protocol safer, when two aircrafts i and j enter an avoid mode, each aircraft determines whether it would be best to make a -90° or a 90° instantaneous turn. Let halfcircle$_i(a_i)$ denote the half-circle made by the i-th aircraft following an a_i-degree instantaneous turn, where $a_i \in \{-90^\circ, 90^\circ\}$. The half-circle halfcircle$_i(a_i)$ is defined similarly. The decision which half-circle to take is based on maximizing the minimum distance between the two aircraft when they follow the half circles with constant angular velocity $\dot{\theta}_i = \textsc{Protocol}(i) = (-\text{sign}(a_i))c$ and $\dot{\theta}_j = \textsc{Protocol}(j) =$

$(-\text{sign}(a_j))c$. This safer protocol eliminates those collisions which could be avoided by making the appropriate -90° or 90° instantaneous turn instead of always turning by -90°, as it is the case in [19, 32, 41]. This safer protocol makes it more challenging to compute witness trajectories.

4.2.2 Experimental Settings

A problem instance is obtained by specifying the number N of the aircrafts, the initial $(x_i^{\text{init}}, y_i^{\text{init}})$, and the goal $(x_i^{\text{goal}}, y_i^{\text{goal}})$ positions for each aircraft i. The experiments carried out in related work [7, 32, 41, 53] relied on one problem instance, where the initial and goal positions were set by hand. In order to test the computational efficiency of `HyDICE` across different problem instances, we use an automatic procedure to generate random problem instances. This allows a more comprehensive testing that better characterizes the computational efficiency of each method. As noted earlier, in the hand-designed problem instance, all the computational methods (`RRT`, `RRT[D`*`]`, `HyDICE[NoLeads]`, and `HyDICE`) solved the problem in less than 10s.

Randomized Problem Instance Generation In the automatic procedure for generating a random benchmark instance, half of the aircrafts are placed from left to right at the top and the other half are placed at the bottom at a safe distance from each other. The aircrafts placed at the top have goal positions at the bottom, and the aircrafts placed at the bottom have goal positions placed at the top. More precisely, let $h = N/2$. The gap between aircrafts is set to $\text{gap} = (2.85 + 0.04 * (N - 10)) * p$, which corresponds to $2.85p$ for $N = 10$; $3.05p$ for $N = 15$; and $3.25p$ for $N = 20$. Then, for each $i = 1, \ldots, h$, which corresponds to the first half of the aircrafts, x_i^{init} is selected pseudo-uniformly at random from $[\text{init}_i, \text{init}_i + p]$, where $\text{init}_i = -500km + (i-1) * \text{gap}$; y_i^{init} is selected pseudo-uniformly at random from $[250, 350]km$; x_i^{goal} is selected pseudo-uniformly at random from $[\text{goal}_i, \text{goal}_i + p]$, where $\text{goal}_i = -420km + (i-1) * \text{gap}$; and y_i^{goal} is selected pseudo-uniformly at random from $[-350, -250]km$. For each $i = h, \ldots, N$, which corresponds to the second half of the aircrafts, x_i^{init} is selected pseudo-uniformly at random from $[\text{init}_i, \text{init}_i + p]$, where $\text{init}_i = -500km + (i-h-1) * \text{gap}$; y_i^{init} is selected pseudo-uniformly at random from $[-350, -250]km$; x_i^{goal} is selected pseudo-uniformly at random from $[\text{goal}_i, \text{goal}_i + p]$, where $\text{goal}_i = -420km + (i-h-1) * \text{gap}$; and y_i^{goal} is selected pseudo-uniformly at random from $[250, 350]km$.

4.2.3 Experiments

Experiments were carried out with $N = 10, 15, 20$ aircrafts, which correspond to continuous state spaces with $30, 45, 60$ dimensions, respectively. For a fixed N, 200 problem instances were generated using the randomized procedure described above in Section 4.2.2. Each method was run on each problem instance. A timeout of 3600s was imposed on each run. The median computational time is reported for each method.

4.2.4 Results

A summary of the results is provided in Table 2. These results indicate the computational efficiency of `HyDICE`. In each case, `HyDICE` is several times faster than `RRT` and `RRT[D`*`]`. As the number of aircrafts is increased, the computational advantages of `HyDICE` become more pronounced.

method	number of aircrafts $N = 10$	$N = 15$	$N = 20$
`HyDICE`	30.35s	42.67s	77.61s
`RRT`	242.15	394.40s	1973.11s
`RRT[D*]`	X	X	X

Table 2 Comparison of the computational efficiency for solving the aircraft conflict-resolution problem with respect to the number of aircrafts N. For each N, the computational efficiency of each method is measured as the median computational time obtained on 200 random instances of the aircraft conflict-resolution problem. Entries marked with X indicate a timeout, which was set to 3600s.

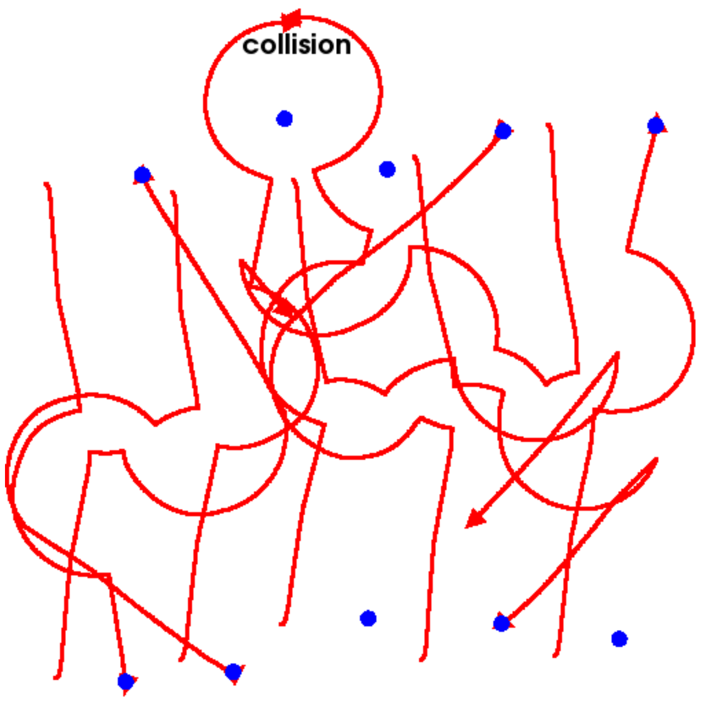

Fig. 4 Example of a witness trajectory that indicates a collision between two aircrafts in a scenario involving 10 aircrafts. Blue circles indicate goal positions.

5 Discussion

We have presented `HyDICE`, a multi-layered approach for hybrid-system falsification that combines motion planning with discrete search. Experiments on nonlinear hybrid systems with numerous modes and high-dimensional continuous state spaces demonstrate the promise of `HyDICE` as a falsification method. Comparisons to related work show computational speedups of up to two orders of magnitude. The combination of motion planning and discrete search in the framework of `HyDICE` raises important computational and theoretical research issues, which are part of ongoing and future investigations.

Scalability in the Discrete Space Although HyDICE was shown to scale up reasonably well and handle a system with slightly over one million modes, the scalability issue remains open to research. As the number of modes becomes significantly large, the graph search used in this work becomes a bottleneck. Methods developed in the verification community, which can handle discrete systems with billions of modes [10], could provide an efficient alternative.

Scalability in the Continuous Space Complex hybrid systems are characterized not only by a large number of modes, but also by high-dimensional continuous state spaces. An important research issue is the improvement of the motion-planning component of HyDICE in order to effectively explore high-dimensional continuous state spaces with hundreds of dimensions. The framework of HyDICE, which focuses on the combination of discrete search and motion planning, opens up the possibility of integrating different motion planners from the one presented in this work, such as RRT [38], EST [28, 50], PDST [35], and others [11, 37].

Low-Dimensional Projections Another issue that arises when dealing with high-dimensional continuous state spaces is the effective computation of low-dimensional projections. The objective of the projection is to reduce the dimensionality, while at the same time preserve the underlying structure of the original dataset. This work relied on simple projections based on specific knowledge about the hybrid systems under consideration.

Toward Increasingly Realistic Hybrid Robotic Systems In robotics applications such as exploration and navigation, which motivated the hybrid-system benchmark in this work, it is often the case that the robotic system should avoid collisions with obstacles. Current work [32, 41, 46] in the context of hybrid-system testing has not considered obstacles. HyDICE can however naturally handle such scenarios. In particular, collision avoidance can be incorporated into HyDICE by considering it as an additional constraint in the invariant that should be satisfied by each state and hybrid-system trajectory that is added to the search tree. Moreover, the presence of obstacles makes it possible to consider other types of decompositions besides the grid decompositions used in this work. In particular, triangular decompositions such as conforming Delaunay triangulations have been widely used in similar settings in computational geometry [16], finite element analysis [22], and robotics [11, 37]. Preliminary results in the context of motion planning show considerable computational improvements when using conforming Delaunay triangulations instead of grid decompositions, and one would expect that similar benefits can be obtained by HyDICE for the falsification of safety properties of hybrid robotic systems that must avoid collision with obstacles at all times.

Acknowledgment

The authors thank Thao Dang for providing implementation details of the related work [41].

References

1. Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, Ho PH, Nicollin X, Olivero A, Sifakis J, Yovine S (1995) The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1):3–34

2. Alur R, Dang T, Ivančić F (2006) Counterexample-guided predicate abstraction of hybrid systems. Theoretical Computer Science 354(2):250–271
3. Alur R, Henzinger TA, Lafferriere G, Pappas G (2000) Discrete abstractions of hybrid systems. Proceedings of the IEEE 88(7):971–984
4. Asarin E, Dang T, Maler O (2002) The d/dt tool for verification of hybrid systems. In: Intl Conf on Computer Aided Verification, LNCS. Springer-Verlag, pp. 365–370
5. Behrmann G, David A, Larsen KG, Mller O, Pettersson P, Yi W (2001) Uppaal - present and future. In: IEEE Conf on Decision and Control, vol. 3. pp. 2881–2886
6. Belta C, Esposito J, Kim J, Kumar V (2005) Computational techniques for analysis of genetic network dynamics. Intl Journal of Robotics Research 24(2–3):219–235
7. Bhatia A, Frazzoli E (2004) Incremental search methods for reachability analysis of continuous and hybrid systems. In: Hybrid Systems: Computation and Control, LNCS, vol. 2993. pp. 142–156
8. Botchkarev O, Tripakis S (2000) Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In: Hybrid Systems: Computation and Control, LNCS, vol. 1790. Springer-Verlag, pp. 73–88
9. Branicky MS, Curtiss MM, Levine J, Morgan S (2006) Sampling-based planning, control, and verification of hybrid systems. Control Theory and Applications 153(5):575–590
10. Burch J, Clarke E, McMillan K, Dill D, Hwang L (1992) Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2):142–170
11. Choset H, Lynch KM, Hutchinson S, Kantor G, Burgard W, Kavraki LE, Thrun S (2005) Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press, Cambridge, MA
12. Chutinan C, Krogh BH (2003) Computational techniques for hybrid system verification. IEEE Transactions on Automatic Control 48(1):64–75
13. Clarke EM, Bierea A, Raimi R, Zhu Y (2001) Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1):7–34
14. Clarke EM, Grumberg O, Peled DA (2001) Model Checking. MIT Press
15. Copty F, Fix L, Fraer R, Giunchiglia E, Kamhi G, Tacchella A, Vardi M (2001) Benefits of bounded model checking at an industrial setting. In: Intl Conf on Computer Aided Verification, LNCS, vol. 2102. Springer-Verlag, pp. 436–453
16. de Berg M, van Kreveld M, Overmars MH (1997) Computational Geometry: Algorithms and Applications. Springer, Berlin
17. Edelkamp S, Jabbar S (2006) Large-scale directed model checking LTL. In: Intl SPIN Work on Model Checking Software, LNCS, vol. 3925. Springer, pp. 1–18
18. Esposito J, Kumar V, Pappas G (2001) Accurate event detection for simulation of hybrid systems. In: Hybrid Systems: Computation and Control, LNCS. pp. 204–217
19. Esposito JM, Kim J, Kumar V (2004) Adaptive RRTs for validating hybrid robotic control systems. In: Workshop on Algorithmic Foundations of Robotics. Zeist, Netherlands, pp. 107–132
20. Fehnker A, Ivancic F (2004) Benchmarks for hybrid systems verification. In: Hybrid Systems: Computation and Control, LNCS, vol. 2993. pp. 326–341
21. Galassi M, Davies J, Theiler J, Gough B, Jungman G, Booth M, Rossi F (2006) GNU Scientific Library Reference Manual. Network Theory Ltd, 2 edn.
22. George PL, Borouchaki H (1998) Delaunay triangulation and meshing : application to finite elements. Hermes Science Publications
23. Giorgetti N, Pappas GJ, Bemporad A (2005) Bounded model checking for hybrid dynamical systems. In: IEEE Conf on Decision and Control. Seville, Spain, pp. 672–677
24. Glover W, Lygeros J (2004) A stochastic hybrid model for air traffic control simulation. In: Hybrid Systems: Computation and Control, LNCS, vol. 2993. pp. 372–386
25. Henzinger T (1996) The theory of hybrid automata. In: Symp on Logic in Computer Science. pp. 278–292
26. Henzinger T, Kopke P, Puri A, Varaiya P (1995) What's decidable about hybrid automata? In: ACM Symp on Theory of Computing. pp. 373–382
27. Henzinger TA, Ho PH, Wong-Toi H (1997) HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer 1:110–122
28. Hsu D, Kindel R, Latombe JC, Rock S (2002) Randomized kinodynamic motion planning with moving obstacles. Intl J of Robotics Research 21(3):233–255
29. Johansson R, Rantzer A (2002) Nonlinear and Hybrid Systems in Automotive Control. Springer, New York, NY

30. Julius AA, Fainekos GE, Anand M, Lee I, Pappas GJ (2007) Robust test generation and coverage for hybrid systems. In: Hybrid Systems: Computation and Control, LNCS, vol. 4416. LNCS, vol. 4416, pp. 329–342
31. Kavraki LE, Švestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans on Robotics and Automation 12(4):566–580
32. Kim J, Esposito JM, Kumar V (2005) An RRT-based algorithm for testing and validating multi-robot controllers. In: Robotics: Science and Systems. Boston, MA, pp. 249–256
33. Kruskal JB (1956) On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society 7(1):48–50
34. Ladd AM (2006) Motion Planning for Physical Simulation. Ph.D. thesis, Rice University, Houston, TX
35. Ladd AM, Kavraki LE (2005) Motion planning in the presence of drift, underactuation and discrete system changes. In: Robotics: Science and Systems. Boston, MA, pp. 233–241
36. Lafferriere G, Pappas G, Yovine S (1999) A new class of decidable hybrid systems. In: Hybrid Systems: Computation and Control, LNCS, vol. 1569. pp. 137–151
37. LaValle SM (2006) Planning Algorithms. Cambridge University Press, Cambridge, MA
38. LaValle SM, Kuffner JJ (2001) Rapidly-exploring random trees: Progress and prospects. In: Workshop on Algorithmic Foundations of Robotics. pp. 293–308
39. Livadas C, Lynch N (1998) Formal verification of safety-critical hybrid systems. In: Hybrid Systems: Computation and Control, LNCS, vol. 1386. Springer-Verlag, pp. 253–272
40. Mitchell IM (2007) Comparing forward and backward reachability as tools for safety analysis. In: Hybrid Systems: Computation and Control, LNCS, vol. 4416. pp. 428–443
41. Nahhal T, Dang T (2007) Test coverage for continuous and hybrid systems. In: Intl Conf on Computer Aided Verification, vol. 4590. LNCS, pp. 449–462
42. Pepyne D, Cassandras C (2000) Optimal control of hybrid systems in manufacturing. Proceedings of IEEE 88(7):1108–1123
43. Piazza C, Antoniotti M, Mysore V, Policriti A, Winkler F, Mishra B (2005) Algorithmic algebraic model checking I: Challenges from systems biology. In: Intl Conf Computer Aided Verification, LNCS, vol. 3576. pp. 5–19
44. Plaku E, Bekris KE, Chen BY, Ladd AM, Kavraki LE (2005) Sampling-based roadmap of trees for parallel motion planning. IEEE Trans on Robotics 21(4):597–608
45. Plaku E, Kavraki LE, Vardi MY (2007) Discrete search leading continuous exploration for kinodynamic motion planning. In: Robotics: Science and Systems. Atlanta, Georgia
46. Plaku E, Kavraki LE, Vardi MY (2007) Hybrid systems: From verification to falsification. In: Intl Conf on Computer Aided Verification, LNCS, vol. 4590. Springer Verlag, pp. 468–481
47. Plaku E, Kavraki LE, Vardi MY (2007) A motion planner for a hybrid robotic system with kinodynamic constraints. In: IEEE Intl Conf on Robotics and Automation. Rome, Italy, pp. 692–697
48. Puri A (1995) Theory of Hybrid Systems and Discrete Event Systems. Ph.D. thesis, University of California, Berkeley
49. Ratschan S, She Z (2007) Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Transon Embedded Computing Sys 6(1):8
50. Sánchez G, Latombe JC (2002) On delaying collision checking in PRM planning: Application to multi-robot coordination. Intl J of Robotics Research 21(1):5–26
51. Silva BI, Krogh BH (2000) Formal verification of hybrid systems using CheckMate: a case study. In: American Control Conference. pp. 1679–1683
52. Stursberg O, Krogh BH (2003) Efficient representation and computation of reachable sets for hybrid systems. In: Hybrid Systems: Computation and Control, LNCS, vol. 2623. Springer-Verlag, pp. 482–497
53. Tomlin CJ, Mitchell I, Bayen A, Oishi M (2003) Computational techniques for the verification and control of hybrid systems. Proc of IEEE 91(7):986–1001
54. Tomlin CJ, Pappas GJ, Sastry SS (1998) Conflict resolution for air traffic management: a case study in multi-agent hybrid systems. IEEE Transactions on Automatic Control 43(4):509–521
55. Yovine S (1997) Kronos: A verification tool for real-time systems. Intl Journal of Software Tools for Technology Transfer 1:123–133
56. Zhang W (2006) State-space Search: Algorithms, Complexity, Extensions, and Applications. Springer Verlag, New York, NY